Day and Night Traffic Light Controller

Utilizing a Timed Implementation

Matt McAuliffe and Emily Boar
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: mmcauliffe@oakland.edu and eboar@oakland.edu

***Abstract*—this paper explores the design of a four-way traffic controller with the options of a day and night mode. Where the night mode consists of flashing yellow for the high traffic road and flashing red for the low-traffic road. It was found that using a clock divider, counters, multiplexers, and finite state machines was the best way to construct the design and implement it on a breadboard.**

I. Introduction

Traffic lights serve an important role in today's society, because they regulate the traffic flow of automobiles. Hence, the motivation towards this project is to gain a better understanding of how a traffic light system can be designed and used on an FPGA board. This project intends to provide a working model of a standalone hardware based traffic controller for a four-way intersection with no left hand turns or right hand turns. It was done by utilizing what was learned in the course about counters, multiplexers, and finite- state machines. Some topics that had to be researched and learned were how to make a clock divider and how to use PMOD ports. This traffic light project brought to light how the system works in real life, as well as how the system could be made on the FPGA using Vivado.

II. Methodology

The knowledge learned in the ECE 2700 course was utilized to design our circuit. First the problem was broken down into smaller steps. After this it became apparent that the problem would be best represented by several states, as a result of this conclusion, a finite state machine was determined to be the most appropriate tool to utilize during the design sequence. To accompany the finite state machine, clock dividers, counters, and multiplexers were used to design the project.

A. Assigning States

Looking into the logic of how a traffic light works, we designed the states. We also decided to include a night and day mode to simulate normal traffic as well as how it would work with flashing lights. Each state in the daytime cycle relies on only the input E to transition it to the next states. For the night cycle it relies on the flashing signal to be active as well as E to be high so that it can transition from a state of being lit to a state where the lights are all off. These states were assigned and are shown in fig. 1. The flashing states are S7 and S8.

A bubble state diagram was also made to understand the flow of the traffic light logic system. E acts as the transition signal between states and each state is labeled to create the led logic value when it is in that state. This state diagram is shown in fig. 2.A bubble state diagram was also made to understand the flow of the traffic light logic system. E acts as the transition signal between states and each state is labeled to create the led logic value when it is in that state. The full state diagram is shown in fig. 2. The flashing states S7 and S8 were put in their own model for neatness and are shown in fig. 3.

Z	State	North/South			East/West		
		G	Y	R	G	Y	R
0 0 0	S1	0	0	1	0	0	1
0 0 1	S2	1	0	0	0	0	1
0 1 0	S3	0	1	0	0	0	1
0 11	S4	0	0	1	0	0	1
1 0 0	S5	0	0	1	1	0	0
1 0 1	S6	0	0	1	0	1	0
1 10	S7	0	1	0	0	0	1
1 11	S8	0	0	0	0	0	0

Figure 1: State table with assignments

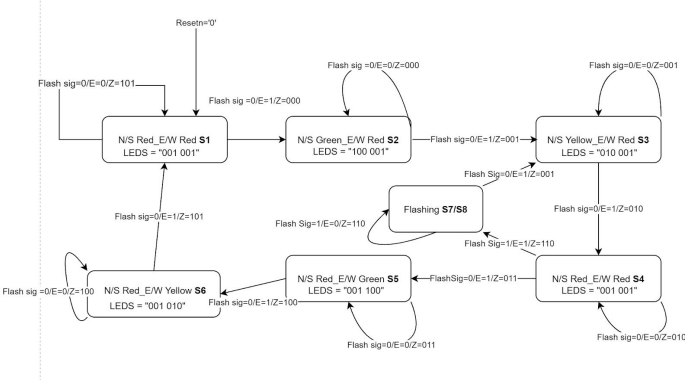

Figure 2: FSM

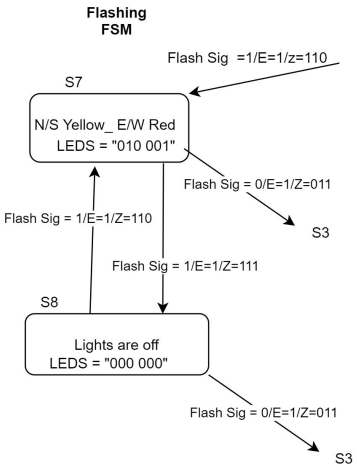

Figure 3: Flashing FSM

B. *Utilizing Clocks*

We decided to manipulate the master clock of 100MHz with one clock divider to create slower time pulses. Originally more clock dividers were designed, but it was realized that they were unnecessary and only one would be needed. This divider would then lead to 5 different counters which will allow the counters to count at a slower pace. The 5 separate counters were set to count for 4 different time intervals. Three of these counters will each take care of two states. For example, S1 and S4 will be controlled by a 40 second counter. The only states that will be controlled by the same amount of time but with different counters will be S7 and S8. These counters will be reset by using the Z value and an encoder to make a five bit reset bus. Where each line is '0' when the corresponding counter is not in use, and '1' for the counter that is currently being used. The clock divider and the five counters and their assigned states are shown in fig. 4.

C. *Selecting with a Multiplexer*

This multiplexer worked just like a regular mux, however some of the clocks were mapped to multiple inputs of the mux since three of the counters are shared by states in the FSM.

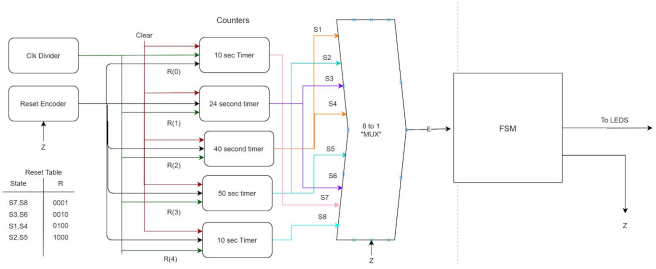

Figure 4: Top diagram of the whole system

D. *Hardware and External Implementation*

Moving from the simulation, it was decided that a breadboard would be the best place to create the final demonstration of the traffic light project. 12 leds and 12 of the 220 ohm resistors along with wires were used. Six leds were hooked up to the pmod port JA on the FPGA, because the rest of the leds would be wired in parallel to those directly hooked up to the JA ports. There were six Leds that simulated traffic flow of the north and south and there were six leds that simulated the traffic of east and west. The resetn state was connected to the CPU reset button on the board. A switch, SW0, was used to simulate the flashing signal in the code, which let the fsm change to night mode or day mode.

III. Experimental Setup

In order to simulate and test the design, a test bench was created. At first the simulation was not running for the expected period of time, so we had to type into the console our desired run time and then it worked. Then it was seen that the simulation was running far too fast, so the integer value in the clock divider was increased. This new longer time was tested physically on the breadboard with the leds since the simulation was taking far long to run. Watching the times of each state through the leds, it was found that the correct integer was used to create a longer more desired time for each state.

IV. Results

When running the test bench with the correct clock divider construction, everything looked like it was working well at first glance. But, upon implementing it on the breadboard, it was realized that the flashing signals were moving far too fast. The issue was debugged and it was noted that another counter had to be made for S8, where all the leds would be off, instead of one counter being put for both S7 and S8. After that add on the project physically ran as expected. The lights all start off as red and then N/S turns green while E/W, then it gradually moved to N/S being yellow while E/W stayed red, and after that the lights all turned red. The cycle then continued in a gradual and safe traffic manor. The physically implemented project can be seen in fig. 5.

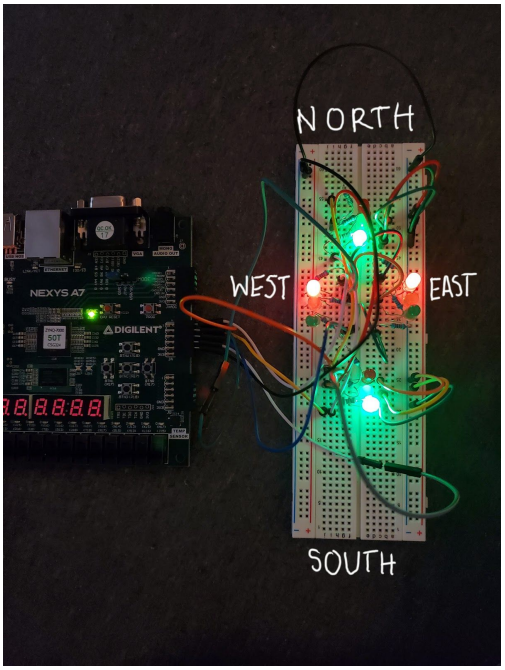

Figure 5: external implementation

CONCLUSIONS

The most significant understanding from doing this project was seeing how the clock can be manipulated and used to control the timing of a finite state machine. There were a few challenges along the way such as understanding how many counters we need along with how the counters would be reset and allow one state to move to the next. The team eventually overcame those challenges and learned from it. One part that had to be learned more externally was how to use the pmod ports for the project. From learning more about that, it taught the team how to wire those pins to provide signals to objects, such as leds, on a breadboard. Some improvements that could be made for a future traffic light system would be to utilize the sensor so that it senses when there are cars waiting at the traffic light so it can change more efficiently. Another improvement would have been allowing for the cars to be able to turn left and right. For emergency vehicles, the sensor could have been used to show that a vehicle is coming and it would give priority right away to whichever direction it was coming from. Despite this traffic light system not having those features, an efficient system was still built and from it the team learned how to create their own design to be coded in Vivado and used by the FPGA board.